Báo cáo thực nghiệm môn Cấu trúc dữ liệu và giải thuật - IT003.Q21.CTTN

Họ và tên: Nguyễn Hải Phong

MSSV: 25521383

Link github: https://github.com/sherwyndd/DSA_sort_project

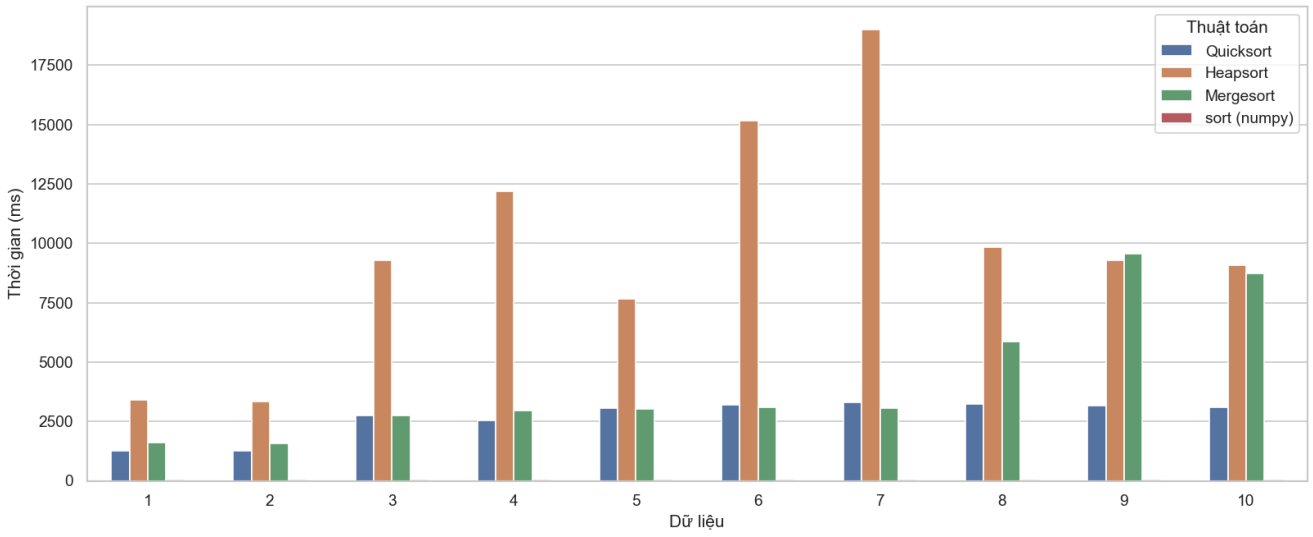
Nội dung báo cáo

I. Kết quả thực nghiệm

1. Bảng thời gian thực hiện

	Thời gian thực hiện (ms)			
Dữ liệu	Quicksort	Heapsort	Mergesort	sort (numpy)
1	1262	3422	1616	40
2	1253	3319	1562	38
3	2737	9285	2764	42
4	2555	12198	2946	38
5	3069	7642	3042	38
6	3205	15144	3110	38
7	3301	19015	3063	36
8	3228	9851	5858	38
9	3173	9289	9559	37
10	3105	9073	8739	38
Trung bình	2689	9824	4226	38

2. Biểu đồ (cột) thời gian thực hiện



II. KẾT LUẬN

Sau khi thực hiện thực nghiệm sắp xếp trên 10 bộ dữ liệu với các kích thước khác nhau, em đưa ra các kết luận sau:

1. Ảnh hưởng của đặc điểm dữ liệu đến thuật toán:

- **Dữ liệu có thứ tự (Test 1 & 2):** Quicksort và Mergesort xử lý rất tốt các dãy đã sắp xếp tăng dần hoặc giảm dần. Tuy nhiên, nếu không được chọn Pivot tối ưu, Quicksort có thể bị suy biến hiệu năng (trong thực nghiệm này kết quả vẫn duy trì ổn định ở mức ~1200ms).
- **Dữ liệu số nguyên và số thực (Test 3 - 10):** Tốc độ sắp xếp số thực (Test 7-10) có xu hướng chậm hơn một chút so với số nguyên do chi phí so sánh và xử lý dấu phẩy động của CPU cao hơn.

2. Đánh giá hiệu năng tổng quan

Quicksort: Là thuật toán có tốc độ xử lý nhanh nhất và ổn định nhất trong các thuật toán tự cài đặt. Thời gian thực thi trung bình duy trì ở mức thấp (khoảng 2,689 ms), chứng minh hiệu quả của tư duy chia để trị và khả năng tối ưu hóa bộ nhớ đệm (cache) tốt.

Mergesort: Đứng thứ hai về tốc độ (trung bình 4,226 ms). Mặc dù có độ phức tạp O(nlogn) tương tự Quicksort, nhưng do chi phí bộ nhớ phụ và thao tác trộn (merge) phức tạp hơn nên thời gian chạy thực tế chậm hơn một chút.

Heapsort: Cho kết quả chậm nhất trong các thuật toán tự cài đặt (trung bình 9,824 ms). Đặc biệt, ở một số bộ dữ liệu lớn, thời gian tăng vọt lên đến 15,000 - 19,000 ms. Nguyên nhân do cấu trúc Heap khiến việc truy cập bộ nhớ không liên tục, làm giảm hiệu năng xử lý của CPU.

Numpy Sort: Có tốc độ vượt trội hoàn toàn so với phần còn lại (chỉ khoảng 38 ms). Điều này cho thấy sức mạnh của việc tối ưu hóa mã nguồn bằng ngôn ngữ C kết hợp với các thuật toán hiện đại (như Timsort) trong các thư viện chuyên dụng.

3. Kiến nghị sử dụng

Trong thực tế lập trình, nên ưu tiên sử dụng các hàm sắp xếp có sẵn của thư viện (như numpy.sort hoặc sort() của Python) để đạt hiệu năng tối đa.

Nếu bắt buộc phải tự cài đặt, **Quicksort** là lựa chọn ưu tiên cho các bài toán ưu tiên tốc độ xử lý.

Mergesort nên được cân nhắc khi cần một thuật toán sắp xếp ổn định (Stable Sort), đảm bảo thứ tự của các phần tử bằng nhau không bị thay đổi.

Heapsort tuy chậm hơn trong thực tế nhưng là giải pháp tối ưu về bộ nhớ khi không muốn tốn thêm không gian lưu trữ phụ (In-place sort).